Shrinking the Genotype: L-systems for EHW ?

Pauline C. Haddow[1], Gunnar Tufte[1] and Piet van Remortel[2]

[1] The Norwegian University of Science and Technology
Department of Computer and Information Science
O.S. Bragstadsplass 2E, 7491 Trondheim, Norway
[2] Vrije Universiteit Brussel
COMO - Department of Computer Science
Pleinlaan 2, 1050 Brussels, Belgium
pauline@idi.ntnu.no, gunnart@idi.ntnu.no,pvremort@vub.ac.be

Abstract. Inspired by biological development where from a single cell, a complex organism can evolve, we are interested in finding ways in which artificial development may be introduced to genetic algorithms so as to solve our genotype challenge. This challenge may be expressed in terms of shrinking the genotype. We need to move away from a one-to-one genotype-phenotype mapping so as to enable evolution to evolve large complex electronic circuits. We present a first case study where we have considered the mathematical formalism L-systems and applied their principles to the development of digital circuits. Initial results, based on extrinsic evolution, indicate that our representation based on L-systems provides an interesting methodology for further investigation. We also present our implementation platform for intrinsic evolution with development, enabling on-chip evaluation of grown solutions.

1 Introduction

The field of evolvable hardware promises many possibilities within optimisation and exploration of new circuit designs apart from one missing factor — scalability. Scalability is the property of a method or solution to keep on performing acceptably when the problem size increases. In our case, acceptable performance may be said to be a non-exponential resource increase and performance decrease. Our problem domain is the evolution of electronic circuits. We have seen in recent years that small electronic circuits may be evolved successfully [1–4]. However, complex circuits are still beyond our reach.

The scalability problem is due to the resource-greedy nature of evolutionary techniques. Generally in EHW, a one-to-one mapping has been chosen for the genotype-phenotype transition. This means that the genotype needs to include all the information required for the phenotype i.e. configuration data to program the device. The larger the circuit to be evolved, the more logic and routing information required. As the complexity of the genotype increases, so increases the computational and storage requirements.

[0] The authors names are given in alphabetical order

A solution to this problem is to shrink the genotype in some way. This may be stated as the genotype challenge, finding new forms of representation for evolving complex circuits. In this work we are interested in investigating ways in which development may be combined with genetic algorithms so as to enable smaller genotypes to evolve complex circuit designs.

Turning to biology, the biological development stages of pattern formation, morphogenesis, cell differentiation and growth (see section 3) provide us with pointers as to the principles that should be included in a design methodology for artificial development. Many alternative methodologies are possible. As a first case study, we consider L-systems [5] which may be said to be based on differentiation and growth through rules for changes and growth. Our goal is to adapt L-systems to developing digital circuits within the constraints of our technology, a virtual EHW FPGA [6] and then map our virtual EHW FPGA to Xilinx Virtex. A genetic algorithm will be used in combination with L-systems to evolve a final solution using intrinsic evolution. The work presented herein is a first attempt at using L-systems in combination with a genetic algorithm to achieve growth on a digital platform.

The paper is laid out as follows. In section 2 we introduce other work in the field and our motivation for selecting the given representation. Since much of this work is focused on biological development we give a brief introduction to biological development in section 3. L-systems are described in section 4. Our virtual EHW FPGA is briefly described in section 5. In section 6, we discuss and present our adaptation of L-systems to digital circuits. Section 7 presents our experimental platform and section 8 presents our initial results. Ongoing and further work is described in section 9.

2 Background and Motivation

To solve the genome complexity issue and enable evolution of large complex circuits, the need to move away from a one-to-one genotype-phenotype mapping is becoming generally accepted. We can expect newer features such as incremental evolution [7],'divide and conquer' [8] or growth [9] to become more common in the work within the field of EHW so as to attack the representation problem.

In the past two to three years, researchers in the field of both software and hardware evolution have begun to look again at biology to gain better insights into improving existing techniques.

Inspired by biological development before differentiation, a new family of fault tolerant FPGAs are being developed at York [10]. Each cell can take over the functionality of a faulty cell as it possesses a complete copy of the "genome". To achieve the degree of flexibility required when any cell can have any function, reconfigurable technology may be said to be a requirement. This is especially the case since the cellular structure varies as a function of the application [11].

Cellular encoding [12] is a well-defined formal approach based on biological development. This approach may be used to build complex systems by encoding

the cell types, timing of cell division and changes in links involving the cell [13] This methodology is a variation of genetic programming [14].

The work of Mjolsness et al [15] focuses on software modelling of morphogenesis in plants to better understand the morphogenesis stage of development. The long term goal of this work is not just single electronic circuits but more towards much more complex systems such as self-sustaining space industry.

One of the reasons for this interest in biology is due to a move from optimisation to exploration [16]. That is from improving existing solutions to finding novel solutions. If we want to explore for novel solutions one approach is to provide *knowledge-poor representations* and give evolution a greater area of freedom to explore for solutions [16]. In a knowledge-poor representation we allow evolution to find solutions that we haven't thought of ourselves. Using knowledge-rich representations we lead evolution to solutions where we think they can be found and in this way limit the search space of evolution to our own specified search space. Using context insensitive L-systems for the representation may be said to be a form of knowledge-poor representation.

To enable growth, one solution is to follow the working of DNA and combine rules into our representation. That is, the genotype includes rules telling how, where and which gene i.e. component in a component representation, should grow to develop a solution [17]. In DNA, instructions fire and suppress other instructions in an increasingly more complex network of activations. L-systems are a mathematical formalism based on this concept.

In this work our goal is to shrink the genotype. Shrinking the genotype effectively moves the complexity problem over to the genotype-phenotype mapping thus increasing the complexity of the mapping. However, in [6], we proposed a solution to this complex mapping problem which includes splitting the mapping process into two stages using a virtual EHW FPGA as the bridge between the genotype and the phenotype. The mapping from the virtual EHW FPGA to a physical FPGA is a simpler mapping as both architectures are based on FPGA principles. The development process itself is the first stage of the mapping. Instead of developing to our complex organism (phenotype) we are developing to a simpler organism the *intertype* i.e. the configuration data for our virtual EHW FPGA.

3 Biological Development

Biological development enables complex organisms to be built in a robust way. What are the features of this reliable system design?

An initial unit, a cell, holds the complete building plan (DNA). It is important to note that this plan is generative — it describes how to build the system, not what the system will look like. Units have internal state, can communicate locally, can move, spawn other units or die. Groups of units may also exhibit group-wise behaviour i.e. a group state.

The global developmental stages from the zygote (fertilised egg) to the multicellular organism, although interdependent and not strictly sequential, may be categorised as *pattern formation; morphogenesis; cell differentiation and growth.*

During *pattern formation* cells are organised in different regions according to their cell types (set of cells with the same gene activity pattern) in order to become distinct parts (body segments) of the eventual organism. One could say that this is a group-wise change of state due to activation of certain codes after local communication or reaction to globally available signals.

Some cells may change shape (expand/contract) exerting a force on other cells. This process is termed *morphogenesis* and is crucial in the formation of general shape in the organism.

The *differentiation* process is where cells become structurally and functionally different from each other. This includes both intra-cellular factors (cell lineage) and inter-cellular interactions (cell induction).

The final step is *growth*. This is the true enlarging of the almost completely formed organism. This is achieved by multiple cell divisions and expansions. During growth, programmed cell death or *apoptosis* can help generate special structures like fingers and toes from continuous sheets of tissue.

4 L-systems

An L-system is a mathematical formalism used in the study of biological development. One of the main application areas is the study of plant morphology. Phenotypes are branching structures attained through the derivation and graphical interpretation of the development process, described by a set of rules. The rules describe how the development should grow and specialise and interpretation of the development itself enables morphogenism to be studied.

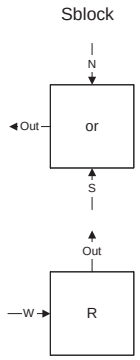
An L-system is made up of an alphabet, a number of ranked rules and a start string or axiom. Applying a rule means finding targets within the search string which match the rule condition. This condition is a pattern on the left hand side (LHS) of the rule. This condition is also a string and the string is made up of elements from the alphabet. Firing the rule means replacing the targets, where possible, with the result of the rule i.e. the right hand side (RHS) of the rule. A more complete description of rule replacement may be found in section 6.2.

Firing of the rules continues until there are no targets found for any rule or until the process is interrupted. In the context of genetic algorithms, the axiom and the associated rules are the genotype and the developed string is the phenotype.

5 Virtual EHW FPGA - Sblock Architecture

The Virtual EHW FPGA contains blocks — *sblocks*, laid out as a symmetric grid where neighbouring blocks touch one another. There is no external routing between these blocks except for global clock lines. Input and output ports enable communication between touching neighbours. Each sblock neighbours onto

sblocks on its 4 sides. Each sblock consists of both a simple logic/memory component and routing resources. The sblock may either be configured as a logic or memory element with direct connections to its 4 neighbours or it may be configured as a routing element to connect one or more neighbours to non-local nodes. By connecting together several nodes as routing elements, longer connections may be realised. A detailed description of the sblock architecture may found in [6] and a discussion around evolution requirements leading to this structure are given in [18].



LUT in the Sblock

LUT Adr.	OR	R
o n e s w	Out	Out
0 0 0 0 0	0	0
0 0 0 0 1	0	1
0 0 0 1 0	1	0
0 0 0 1 1	1	1
0 0 1 0 0	0	0
0 0 1 0 1	0	1
0 0 1 1 0	1	0
0 0 1 1 1	1	1
0 1 0 0 0	1	0
:		
1 1 1 1 0	1	0
1 1 1 1 1	1	1

Fig. 1. Sblock Configuration : An OR gate (OR) and Routing from the West (R)

The main element of an sblock is its 5 input LUT. Configuration of the sblock consists of configuring the response to the inputs of the LUT. With a 5-input LUT, 32 configuration bits are required. The sblocks are programmed as either logic or routing blocks. Figure 1 illustrates the response for a logic — two input OR gate (OR), and a routing — west router (R) sblock. For logic, the output response programmed reflects the active inputs and the functionality of the LUT. Looking at the LUT contents it can be seen that the OR gate of the north and south inputs is achieved by ignoring the west and east inputs. For routing, only one input is active, the others are don't care. In the example given, a west routing sblock is indicated as the response matches the west input.

The configuration string of the sblock architecture consists of all the sblock configurations (32 bits), starting at the top left hand corner and crossing the grid a column at a time. That is, for a 16x16 grid, the configuration string will consist of 32x16x16 bits.

6 Development

As stated in section 2, a knowledge-poor representation may free evolution to explore for solutions in a large search space. Our interpretation of knowledge-

poor in an L-system context, is that the rules do not have any chosen meaning with respect to improving connectivity or logic. Also the rules are not context-sensitive, relying on neighbouring sblock information. During evolution, the rules themselves evolve.

The application of the rules to the developing intertype has been designed with determinism in mind. As such, the individual representation provides a reliable representation of the resulting phenotype.

We have added technology constraints where our intertype technology is the virtual EHW FPGA. Following the principles of biological development and treating an sblock as a cell, then growth steps are limited to 32 bit sblocks i.e. a complete cell. As any other developing organism we wish to introduce shape and our goal is the shape of an sblock architecture — a grid. Therefore growth is limited to the grid size chosen for our virtual EHW FPGA. Change rules both effect connectivity of the architecture — in biological terms communication between cells, as well as functionality — specialisation.

6.1 Change and Growth Rules

There are two types of rules : change and growth rules. Figure 2 illustrates part of the rule list used in the experiments of section 8. Change rules have a RHS string of equivalent length to their LHS string. This is to avoid any growth due to the application of a change rule. These rules are ranked from the most to the least specific, as shown. The growth rules are given a random priority and ranked accordingly. In this example change rules have been ranked before growth rules. It may be noted that since these rules are random generated and then evolved, more than one rule may have the same LHS. A don't care feature, typical of digital design, has been introduced to the change rule concept. Don't cares are represented by a 2, as shown. To retain determinism don't cares are only allowed on the LHS of a change rule.

```
1  - 21200110101212 -> 10110100001110
2  - 000012110111   -> 100001110011
3  - 010121121222   -> 100100010100
4  - 2220021022     -> 0111001110
5  - 01001 ->  00101110011101100110000000100111
6  - 10111 ->  10010111000111010101000110010110
7  - 01011 ->  10101100111111011101100100011100
8  - 10101 ->  01011110111001100100001011101010
9  - 00011 ->  11111011001000011001011001101010
10 - 01011 ->  10111101001010010011011110000101
11 - 01110 ->  11010001001000110101011001000001
12 - 01100 ->  11111100101100111000101010111001
```

Fig. 2. Change and Growth Rules

Through change rules, the contents of one or more sblocks are changed. A change rule targets the locations in the intertype where the string on the LHS matches. Firing the rule means replacing this string at the different targets with

the RHS of the rule. The LHS string may be found within a single sblock or overlapping two or more sblocks as illustrated in figure 3.

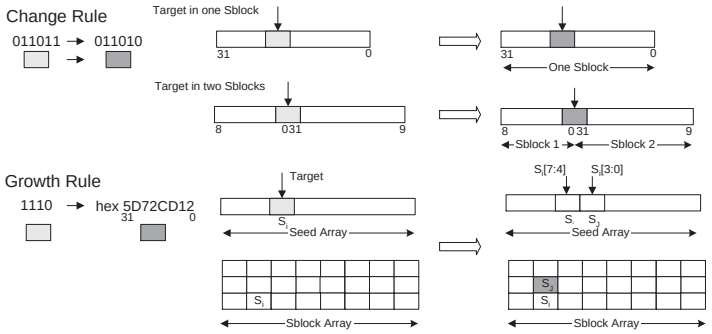


Fig. 3. Applying Change and Growth Rules

As illustrated in figures 3, each sblock has an associated seed. If a seed in the array matches the LHS of a growth rule then this means that the associated sblock is a target for the growth rule. If there is no available space to grow into i.e. the sblock is surrounded by configured sblocks, then there will be no growth at this point. Firing the growth rule means placing an sblock in the first free location according to the priority rule : north, south, west and then east. In introducing a new seed it is important that determinism is maintained. Therefore, the new seed is given the first 4 bits of the sblock which was targeted and its bits 5 to 8 are used to replace its own seed. As shown, the new sblock is configured by the RHS of the growth rule.

6.2 Rule Firing Sequence

To ensure fairness, we propose firing the rules in batches. The rules are still ordered from the most to the least specific but all rules will have a chance of firing before a given rule can fire again.

Figure 4 illustrates this firing sequence. Once any rule in the batch has fired on a target then no other rule can fire on the same target area. As such, these target areas are reserved for the remainder of the current firing batch, as illustrated in what is termed the update string. The default value of the update string is the configuration of the input and output sblocks.

We assume, as shown, that rule 1 (the top ranked rule) has found targets in the intertype i.e. matching strings to its LHS. However, checking the update string, one of its targets can't fire as the target overlaps an input sblock. The rule fires on the other targets in the normal way by writing its RHS string over the target bits at each target point. In addition, it sets the equivalent bits in

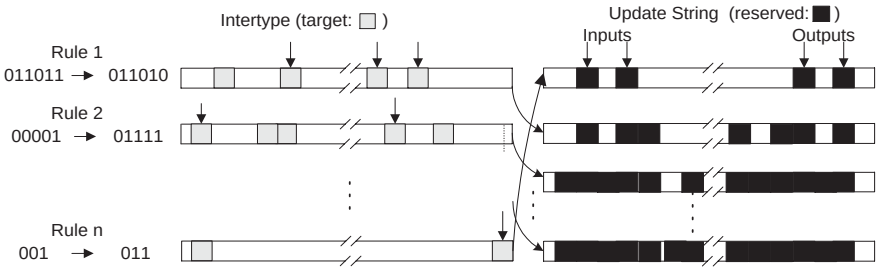


Fig. 4. Rule Firing Sequence

the update string. Rule 2 finds 5 targets but only 2 are free. Again the rule fires and sets the respective bits in the update string. The last rule of the ranked list finds one target and therefore can fire on that target. The update string can now be reset to its default value and the firing process continues beginning again with the most specific rule. In the example illustrated in figure 2, growth rules are ranked after change rules and therefore don't effect the firing of the change rules within a given batch. However, their effect will be seen at the next batch where the update string will reflect the size of the grown intertype. The firing process may be stopped either when there are no more rules to fire or when the developed string meets some other chosen condition.

7 Experimental Platform

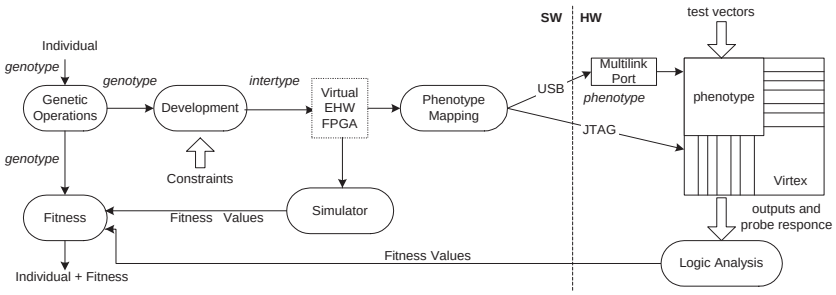


Fig. 5. Platform for Intrinsic Evolution with Development

Figure 5 illustrates the workings of our experimental platform. The genetic algorithm (GA) is implemented using GAlib. Our genotype may consist of one or

more 32 bit axioms and a ranked list of rules. The GA works on a population of individuals where each individual consists of one or more axioms and the ranked list of rules. Genetic operations applied to individuals affect both its axiom(s) and its list of rules.

The individual is then placed in the fitness module in its genotype form, as shown. To start the evaluation phase, the individual is sent to the development process.

The development process is given the constraints of the required design. These include the technology constraint — sblock grid size, and the design features — the number and position of inputs and outputs. The sblocks specified in the design features are protected from further changes during the development process. The axiom(s) is allocated to an sblock location(s). The development process may then begin firing rules, according to the method described in section 6.2 and, as such, develop the genotype to the intertype string.

A simulator is available to generate fitness values based on the intertype representation. This simulator enables initial studies of the interplay between the L-system rules and the GA to be conducted at the intertype level. Fitness values generated are fed back to the fitness module.

The next stage involves mapping the virtual sblock FPGA design (intertype) to a circuit description for Xilinx Virtex FPGA (phenotype). The phenotype mapping is a translation process from the configuration data expressed in the intertype, to Virtex configuration data format. The intertype and the phenotype are configured in columns. In the intertype each sblock's 32 bits is a continuous string, enabling interpretation and growth of the sblock architecture in 32 bit blocks. However, in the Virtex configuration, configuration data is split into frames where configuration of a single CLB is split over several frames. Mapping involves translation between these two formats. In addition, all outputs from the mapped sblock grid are routed to their respective edges so as to be available on the Virtex pads.

To down-load the configuration data to our Virtex chip we have tested out two solutions using Xilinx interfaces. The first is a serial configuration using the USB port of the PC, through Xilinx's multilink port. The second, which is slightly faster uses a JTAG parallel cable.

For fitness evaluation we need feedback of fitness values from the implemented phenotype. The fixed location of the inputs and outputs during the development process means that test vectors may be applied and responses monitored. A number of internal probes on the Virtex chip are also available to provide additional information for fitness evaluation. Responses are fed through the logic analyser, to generate fitness values for fitness evaluation.

8 Experimentation and Evaluation

The experimental goal may be expressed as follows. The "circuit" has no specified function. From a single axiom in the centre of the sblock grid, an sblock solution

consisting of north, south, west or east routing modules is grown. The maximum grid size is 16 by 16.

The experiments were conducted using the development platform in extrinsic evolution mode. That is, fitness values were generated from the simulator and not from the Virtex chip itself. The GA parameters were as follows: population 300; roulette wheel selection with elitism; crossover 0.8 or 0 (2 cases); mutation 0.1 and maximum number of generations 150.

Although our goal is to evolve routing sblocks, fitness evaluation is also provided information about how close non-routing blocks are to routing blocks. That is low-input sblocks are credited more than high input sblocks except for the case of zero-input sblocks. These are given a low weighting as they do not offer any routing possibilities. Fitness is expressed in equation 1.

$$F = 15*R+[6*(C-R)-(6*6in+5*5in+4*4in+3*3in+2*2in+1*N1in)] \quad (1)$$

where R is the number of routing sblocks; C the number of configured sblocks; Xin the number of sblocks with X inputs and $N1in$ the number of inverter sblocks. A pure router sblock has only one input and no inverter function.

Little growth was achieved in the initial experiments before the rules stopped firing. To aid growth and ensure that firing continued, we both increased the ratio of growth to change rules and made the change rules less specific i.e. shorter LHS. A significant improvement in growth was seen but the results presented no clear trend and seemed almost random.

Studying the results more closely, it was obvious that the present setup had inherent epistasis properties. To further test our assumptions, we removed the crossover operator which we believed was forcing the results to jump around in the fitness landscape. As a result the population gradually found better and better solutions and the average fitness improved gradually. Figure 6 illustrates one of our runs where a typical pattern of fitness improvement is seen. Decreasing number of inputs is illustrated from dark gray to white. White boxes with a cross are router modules. Black boxes indicate either zero-input sblocks or non-configured sblocks i.e. not grown. In general our solutions achieved 30 to 50% routers in less than 100 generations. Limitations in our current simulator have not allowed us to run enough generations to study further improvements.

The achievement of a trend towards increasing routing modules indicates that the combination of the GA and L-systems can at least approach a solution, although the solution achieved may be said to be closer to hill climbing than evolution.

We discussed in section 2 the idea of a knowledge-poor representation. However, our experimental results indicate that our current representation might not be as knowledge-poor as we intended. Figure 6 shows that most router blocks achieved are west router blocks, indicating that west router blocks were favoured over other router blocks. In an earlier set of experiments where zero-inputs were omitted from the fitness calculation, zero-inputs were favoured over the poorly weighted high-input blocks resulting in grids with a large number of zero-inputs.

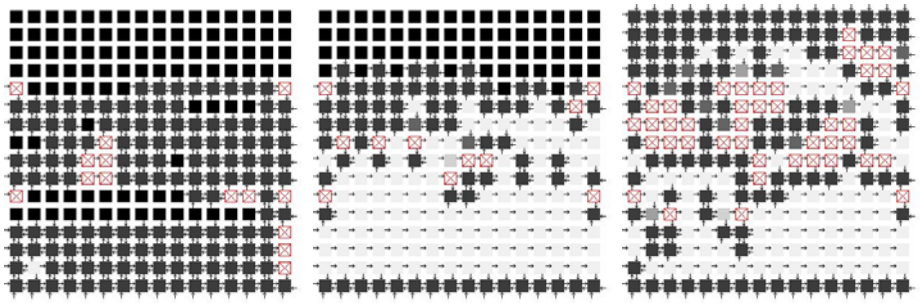


Fig. 6. Configured sblocks after 3, 23 and 57 generations

In both cases, our fitness function did not give special credit to the favoured configurations. As such, this tendency towards a specific type of sblock must lie in our representation i.e. in our rules

9 Ongoing and Future Work

As stated, the goal of our work is to shrink the genotype by using some form of growth to an intertype representation and map to a phenotype representation for fitness evaluation. The platform status is that we can test out individuals and obtain detailed feedback for fitness evaluation. However, running generations for a reasonable population is limited due to the slow Virtex interface. For intrinsic experimentation a better interface is essential and is the focus of our attention at present. The new interface will have the advantage that it will allow us to take more control over placement and routing.

We have presented our representation as an adaptation of L-systems. A number of assumptions have been made based on how the development might proceed and how the many different factors might interact. Issues such as, to name but a few: how many axioms? resolving overlapping growths, where to grow from on the chip, how to apply crossover to rules, seed size required and many more are unresolved. As such, we are not in a position to say whether this representation will lead to the possibility to evolve complex circuits. As such, much experimentation is needed to investigate this representation further. However, intuitively we feel that an L-system based representation is worth further investigation.

References

1. A. Thompson. Evolving electronic robot controllers that exploit hardware resources. In *The 3rd European Conference on Artificial life (ECAL95)*, 1995.
2. A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *1st International Conference on Evolvable Systems, ICES*, Lecture Notes in Computer Science, pages 390–405. Springer, 1996.

3. J.R. Koza et al. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2):109–128, July 1997.
4. T. Higuchi et al. Evolvable hardware and it's application to pattern recognition and fault-tolerant systems. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware : the Evolutionary Engineering Approach*, volume 1062 of *Lecture Notes in Computer Science*, pages 118–135. Springer, 1996.
5. Aristid Lindenmayer. Mathematical Models for Cellular Interactions in Development. *Journal of Theoretical Biology*, 1968.
6. P. C. Haddow and G. Tufte. Bridging the genotype-phenotype mapping for digital FPGAs. In *to appear in the 3rd NASA/DoD Workshop on Evolvable Hardware*, 2001.
7. I. Harvey et al. Why evolutionary robotics? *Robotics and Manufacturing: Recent Tends in Research and Applications*, 6:293–298, 1996.
8. J. Torresen. A divide-and-conquer approach to evolvable hardware. In *2nd International Conference on Evolvable Systems, ICES*, Lecture Notes in Computer Science, pages 57–65. Springer, 1998.
9. P. J. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pages 35–43, 1999.
10. C. Ortega and A. Tyrell. A hardware implementation of an embyonic architecture using virtex FPGAs. In *Evolvable Systems: from Biology to Hardware, ICES*, Lecture Notes in Computer Science, pages 155–164. Springer, 2000.
11. L. Prodan et al. Biology meets electronics: The path to a bio-inspired FPGA. In *Evolvable Systems: from Biology to Hardware, ICES*, Lecture Notes in Computer Science, pages 187–196. Springer, 2000.
12. F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Journal of Evolutionary Computation*, 1993.
13. H. Kitano. Building complex systems using development process: An engineering approach. In *Evolvable Systems: from Biology to Hardware, ICES*, Lecture Notes in Computer Science, pages 218–229. Springer, 1998.
14. J. Koza. *Genetic Programming*. The MIT Press, 1993.
15. E. Mjolsness et al. Morphogenesis in plants: Modelling the shoot apical meristem and possible applications. In *The first NASA/DoD Workshop on Evolvable Hardware*, pages 139–140, 1999.
16. P.J. Bentley. Exploring component-based representations - the secret of creativity by evolution. In *Fourth Intern. Conf on Adaptive Computing in Design and Manufacture*, 2000.
17. P.J. Bentley and S. Kunar. Three ways to grow designs: A comparison of embryongenies for an evolutionary design problem. In *GECCO*, pages 35–43, 2000.
18. P.C. Haddow and G. Tufte. An evolvable hardware FPGA for adaptive hardware. In *Congress on Evolutionary Computation(CEC00)*, pages 553–560, 2000.